Numerical algorithm for simulations of Brownian particles in an ionic channel with fixed-density boundary conditions

Oscar Escolano, Eric Planas
(Dated: May 31, 2019)

We learn a new programming language, Fortran, in order to use it to simulate a system of many particles that follow Brownian dynamics. In particular, we have modelled an ionic channel under the action of different potentials and we have analysed the ion density profile during both transient and steady states and compared it with the theoretical solution of the Fokker-Planck equation. The algorithm includes the temporal evolution of each individual particle through the integration of an stochastic differential equation and the implementation of fixed-density boundary conditions.

I. INTRODUCTION

This project has two main objectives. The first one is to learn Fortran, a new programming language for us, and the second one is to apply it to do our first simulation of a complex system. Specifically, we have simulated an ionic channel. This kind of structures span the cell membrane and are very relevant in cellullar physiology, since thanks to the fast ion flows through them, they allow the neurons to depolarize and transmit the action potential. It is interesting, then, to develop and algorithm that allows us to study this system under different membrane potentials and ion concentrations in and out of the cell in order to better understand its dynamics.

II. THEORETICAL FRAMEWORK

Our approach to simulate the system will be based in the proposed one by our tutor Laureano Ramirez de la Piscina in his work [1]. Hence, we will consider a set of independent particles that move along the channel. It will be done, however, a translation of the problem to one-dimension provided that the system has a paraxial symmetry and the electric field that appear is parallel to the channel axis. We will also consider that the dynamics of each particle in a channel of length L will follow the Langevin equation. Considering a small Reynolds number, it can be assumed the overdamped limit and use the first order equation

$$\gamma \frac{dx}{dt} = -\frac{dV(x)}{dx} + \xi(t) \tag{1}$$

Where ξ(t) is a Gaussian noise with zero mean and a standard deviation $\sqrt{2k_BT\gamma}$ that will simulate the thermal fluctuations and other stochastic forces that may act on the particles, γ is the friction coefficient and V is a time-independent potential that will simulate the membrane potential. Regarding the boundary conditions, we will consider that at both sides of the channel there are two particle reservoirs with a given fixed values of densities, since the flow of ions in and out of the channel will suppose minimum changes in the values of the concentration of this ion in and out of the cell. Once we obtain the results of the simulations applying the above-mentioned model, we will compare them with the Fokker-Planck equation for concentrations as a theoretical reference

$$\frac{\partial \rho(x,t)}{\partial t} = -\frac{\partial}{\partial x} f(x)\rho(x,t) + D\frac{\partial^2}{\partial x^2}\rho(x,t) \tag{2}$$

where $f(x)$ and D are given by

$$f(x) = \frac{1}{\gamma}\frac{dV(x)}{dx}, D = \frac{k_BT}{\gamma} \tag{3}$$

III. NUMERICAL ALGORITHM

In order to simulate the system, we have written a code that will take into account different items such as the temporal evolution of the particles, the implementation of the boundary conditions and the density calculations that will be explained below. All of them can be checked in the code in Appendix A.

A. Time-discretized Langevin dynamics

In order to implement the simulation of the particle motion in an ionic channel, many aspects have to be taken into account in parallel. Firstly, it must be considered the time evolution of the system. As stated above, we will use an Euler algorithm to integrate the overdamped Langevin equation. For this, a temporal step Δt has been set, and the particle positions at the end of each time-step will be calculated using the previous position, obeying

$$x(t+\Delta t) = x(t) + f[x(t)]\Delta t + \frac{1}{\gamma}\sqrt{\Delta t}\ \xi(t) \tag{4}$$

Where ξ is a Gaussian random number with zero mean and standard deviation $\sqrt{2k_BT\gamma}$.

B. Boundary conditions

As mentioned, fixed-density boundary conditions will be considered in order to study the dynamics at the channel boundaries. We will consider that at the end of each time step, the particles that have moved to a position out of the limits, that is if x is smaller than 0 or larger than L, will be removed from the system. That means that the boundaries will have no memory about the previous time iterations. To implement that, every time that a particle exits, its information must be replaced by the data of the last particle in the data vector that is still located in the channel.

On the other hand, at any time step it must be also considered the possible entrance of particles in the channels. For the particles entrance, not only the rate at which a particle can enter has to be studied but also the position where it may enter. The mean number of particles that enter the channel in a time Δt will be

$$\langle N \rangle = \rho \sqrt{D \Delta t}\ q(a) \tag{5}$$

Where a and the function $q(x)$ are given by

$$a = -f \sqrt{\frac{\Delta t}{4D}} \tag{6}$$

$$q(x) = -x\ erfc(x) + \frac{1}{\sqrt{\pi}} e^{-x^2} \tag{7}$$

Note that the rate of entering particles will depend on the potential value at the boundary and the densities of the reservoirs. The number of particles that enter the channel each time should be computed from a Poisson distribution with the calculated mean number $\langle N \rangle$ since each particle arrival is an independent event with respect to the arrivals of other particles. However, with a small time step and not very large densities it can be seen that $\langle N \rangle \ll 1$. In this case the probability of appearing one single particle in a time Δt can be written as $\rho^{(1)} \approx \langle N \rangle$. Then, in the algorithm it will only be necessary to generate a uniform random number $\chi \in (0, 1)$ for each boundary and compare it with $\rho^{(1)}$. If $\chi \leq \rho^{(1)}$ then a particle will enter. Otherwise, no particle enters.

In the case that in a certain time-step a particle enters, it should be decided at which position it is spawned. They cannot be placed just at the edge of the channels since due to the Brownian motion in very few iterations they would exit the channel again. To do this, the cumulative distribution function for the positions of entering particles has been used:

$$F(x) = 1 - \frac{q(\frac{x - f\Delta t}{\sqrt{4D\Delta t}})}{q(a)} \tag{8}$$

Then generating a uniform random number for each new particle, we get the value of F and we obtain the desired position, which will be given by

$$x = F^{-1}(x) \tag{9}$$

In our case, at the beginning of the program, using Newton method, a library with many values of x and $F(x)$ is created and whenever is needed the inverse function is computed via interpolation. There is, however, one exception, for large x, where to compute the corresponding x the following approximation is used the following expression:

$$x \approx f\Delta t + \sqrt{-4\gamma k_B T \Delta t \ln\left(\sqrt{\pi}\ q(a)(1-\chi)\right)} \tag{10}$$

Note that taking into account the possible drifts at the boundaries, we expect that the rate of entering and leaving particles in the boundaries at steady-state is constant, so the ion density near the boundaries should be the same than the densities in the corresponding reservoirs.

C. Density calculation

To calculate the particle density profile in the channel at a given time, we have made an algorithm that divides the channel in 500 bins and the number of particles in each bin at each time step is counted. As stated, our code uses several approximations provided that it is being considered that the densities both in the boundaries and inside the channel are not large and the time step is small enough. With these parameters, however, the measured densities are expected to present large fluctuations. In order to reduce the dispersion in the results it is convenient to average the results by running many independent realizations of the system reducing dispersion by $1/\sqrt{N}$, being N the number of independent realizations. This can be done without problems since the particles are independent and noninteracting.

D. Random Gaussian distribution generation

To generate the Gaussian random variables we use the Box-Müller transform on a uniform random variable. To generate the uniform random variable we use a linear congruential generator for a seed of 8 bytes of memory. At each realization the seed is updated, giving new pseudo-random numbers. [3]

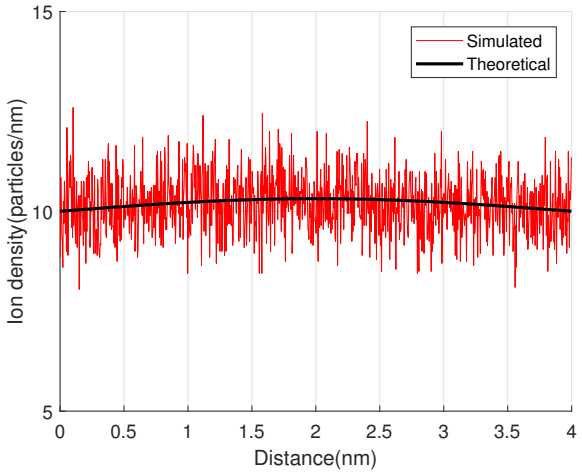

FIG. 1. Particle concentration vs position in the steady state for a null external potential and $\rho_1 = \rho_2 = 10$. Black line: numerical solution of Fokker-Planck equation under these conditions. Red line: Average over 10000 independent realizations in a temporal window $T = 10\ \mu s$. It can be seen that the system almost reaches steady-state.

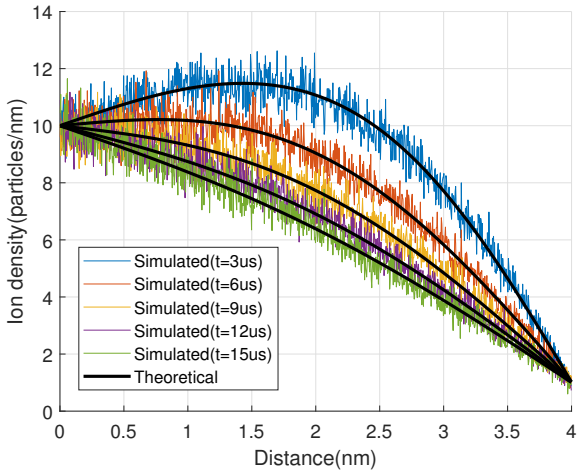

FIG. 2. Temporal evolution of particle concentration vs position until reaching steady-state for a null external potential and $\rho_1 = 10$ and $\rho_2 = 1$. Black line: numerical solution of Fokker-Planck equation under these conditions. Coloured lines: Average over 10000 independent realizations at different times.

IV. RESULTS

In order to study the temporal evolution and the steady-state of this system, we have considered several cases with different potentials and densities at the boundaries. These different conditions will be set by changing the corresponding parameters in the code.

In every case the same initial conditions have been set. That is, at the beginning of the simulations, there will be 50 uniformly distributed particles inside the channel. That means the initial density profile will be constant with $\rho = 12, 5$.

We have performed several simulations of Brownian particles in a channel of the following characteristics: The length has been set to $L = 4$, $\gamma = 1000$, and the chosen temperature has been $k_B T = 25$. We have set the system of units based on length 1 nm, energy 1 meV and time $0,1\ \mu s$. We have also run the simulation during enough time for the system to be close to its steady-state.

As stated before, all the simulation results will be compared with the numerical solution of Fokker-Planck equation, which will be painted in black. In Appendix B it is shown the code for its numerical solving.

We first consider that there is no membrane potential in order to study the particle motion only taking into account the Brownian dynamics. In the first run of the simulation, we have considered that there is no drift ($V = 0$) and the ion concentrations at both sides of the channel is the same ($\rho_1 = \rho_2 = 10$) (Fig. 1.). It can be appreciated that the result close to the steady state is a constant density profile that converges to the theoretical solution. It can be appreciated, however, that under these conditions the noise in the density calculation is quite strong and more realizations may be needed to reduce it.

We have also run a simulation maintaining the null drift but changing the particle concentration at the boundaries $\rho_1 = 10$ and $\rho_2 = 1$. The results are shown in Fig. 2. where it can be seen that the particle density at the steady-state describes a linear profile, which also converge to the theoretical solution.

In the following simulations there have been also considered several different boundary conditions but now a constant drift has been added ($V = (q\phi/L)x$), simulating an standard membrane potential. In circumstances where the particle density at the boundaries is the same, a constant flow of particles is expected, so the density profile would be the same as in the case for null potential. We have decided, however, to simulate a more realistic situation where the concentrations in and out of the cell are different. This would correspond to a situation where the channel is open and the ion flow through it is governed by the electrochemical gradient. In Fig. 3 it is shown the temporal evolution of the system when $\rho_1 = 10$ and $\rho_2 = 1$. In Fig 4 these conditions have been exchanged ($\rho_1 = 1$ and $\rho_2 = 10$). In both cases it can be observed that at every time the solutions of the simulation converge to the theoretical solution.

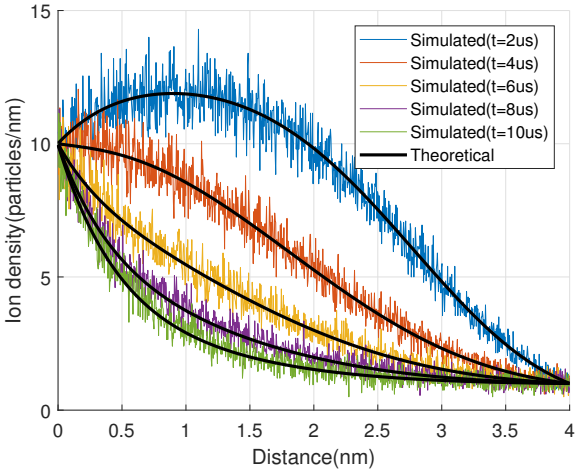

FIG. 3. Temporal evolution of particle concentration vs position until reaching steady-state for a linear external potential with $q\phi = 8k_BT$ and $\rho_1 = 10$ and $\rho_2 = 1$. Black line: numerical solution of Fokker-Planck equation under these conditions. Coloured lines: Average over 10000 independent realizations at different times.

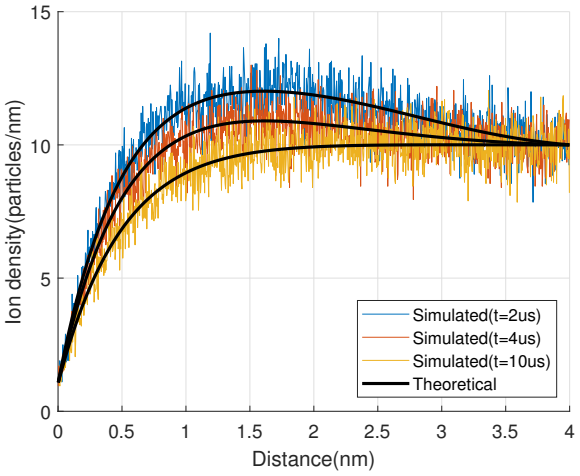

FIG. 4. Temporal evolution of particle concentration vs position until reaching steady-state for a linear external potential with $q\phi = 8k_BT$ and $\rho_1 = 1$ and $\rho_2 = 10$. Black line: numerical solution of Fokker-Planck equation under these conditions. Coloured lines: Average over 10000 independent realizations at different times.

Finally, we have considered how to model the shape of a gate in the ionic channel. This problem is intrinsically a third-dimensional problem, since it involves the three-dimensional structure of the proteins that form the channel. Nevertheless, an equivalent projection to a one-dimensional problem can be done by adding a barrier potential. The shape of the barrier can be very diverse, but we, following

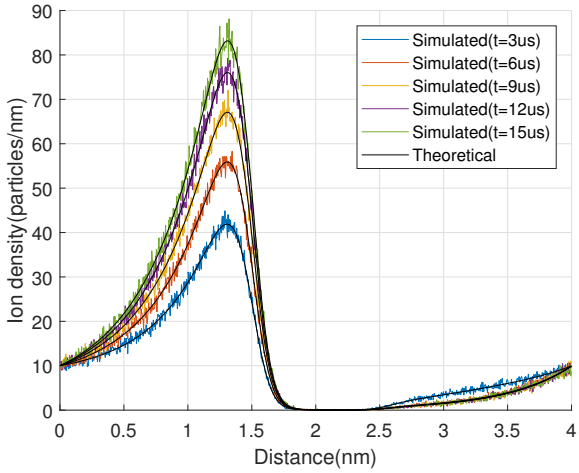

FIG. 5. Particle concentration vs position during the transient with a linear external potential plus a gaussian barrier specified in Eq.(11) and boundary densities $\rho_1 = \rho_2 = 10$. Black line: numerical solution of Fokker-Planck equation under these conditions. Coloured lines: Average over 10000 independent realizations at different times. It can be seen how particles accumulate in the barrier and can barely pass, as if the channel gate was closed.

Ref [2], have opted for adding a Gaussian barrier to a constant drift. That is

$$V(x) = \frac{q\phi}{L}x + V_b e^{-\frac{(x-x_b)^2}{2l^2}} \tag{11}$$

Where V_b is the height, l is the width and x_b is the centre position of the barrier. We have taken the values $V_b = 8k_BT$, $l = L/16$ and $x_b = L/2$. In Fig. 5 the temporal evolution of this system is shown. It is observed again how the simulations agree with the Fokker-Planck simulations.

V. CONCLUSIONS

We consider that the main goals of these project have been achieved: We are now fluent in programming in Fortran and we have managed to obtain the expected results in our simulations. We have been able to translate the problem into a numerical algorithm that, as checked, works well and fulfills the theoretical solutions of the Fokker-Planck equations. In order to go further in this project, it would also have been interesting to program the full three-dimensional problem. Nevertheless, this implies the implementation of more complex boundary conditions in y and z directions, for which we would need more time. However, as the obtained results in the one-dimensional problem have been satisfying, we consider that having reached that point the presented project is complete by itself.

References

[1] L Ramírez-Piscina. "Fixed-density boundary conditions in overdamped Langevin simulations of diffusion in channels". In: *Physical Review E* 98 (Apr. 2018). DOI: 10.1103/PhysRevE.98.013302.

[2] L Ramírez-Piscina and J Sancho. "Physical properties of voltage gated pores". In: *The European Physical Journal B* 91 (Apr. 2018), p. 10. DOI: 10.1140/epjb/e2017-80569-5.

[3] W.T.Vetterling W.H.Press S.A.Teulosky and B.P.Flannery. *Numerical Recipes: The Art of Scientific Computing*. New York: Cambridge University Press, (2007).

Appendix A Simulation code

```
program simulation

!Variable declaration:

integer*8 :: i, nm, ns0, nsF, ne0, neF, nrealitzacions, ne0n,   neLn
integer*8 :: gg, itt, it1, it2, it3, it4, ipos
real*8 :: N1, N2, rho1, rho2, D, kbt, gamma, dt, a1, a2, f1,   f2
real*8 :: Length, pot, derpot, qfun, arg, qa1, qa2, Fx, pi,   Fxv
real*8 :: suma2, deltax
real*8 :: x0, y, xn, xre, newton, m, xentrada, dum, a, b, c,   ns0n
real*8, dimension(0:9999) :: vFxL, vFy0, vFyL,vFx0
integer*4 :: seed
parameter (numbin=1000)
parameter (nparticmax=1000)
parameter (Length=4.0)
parameter (dt=5.0E-4)
real*8, dimension(nparticmax) :: xpos
real*8, dimension(0:numbin-1) :: density, density1, density2
real*8, dimension(0:numbin-1) :: density3, density4, density5
real*8, dimension(0:numbin-1) :: densityt1, densityt2, densityt3
real*8, dimension(0:numbin-1) :: densityt4, densityt5
dimension nbin(0:numbin)
real*8, dimension(nparticmax) :: soroll
real*8 :: U1, U2, r8_uniform_01, r8_normal_ab, nsLn
parameter (kbt=25.d0, gamma=1000.d0, D=kbt/gamma)
parameter(rho1=10.d0, rho2=10.d0)
parameter(itt=300000,it1=itt/5, it2=2*itt/5)
parameter(it3=3*itt/5,  it4=4*itt/5)
common pi, seed
pi=4.d0*atan(1.0_8)
seed=933543
print*,'Initial␣number␣of␣particles␣=␣', nparticulas
deltax=Length/float(numbin)!Bin length


!Boundary conditions

f1=-derpot(0.d0)/gamma
f2=-derpot(Length)/gamma
print*,'pot␣0␣=␣', pot(0.d0)
print*,'pot␣2␣=␣', pot(2.d0)
print*,'pot␣L␣=␣', pot(4.d0)
print*,'derpot0␣=␣', derpot(0.d0)
print*,'derpotL␣=␣', derpot(Length)
a1=-f1*sqrt(dt/(4.d0*D))
N1=rho1*sqrt(D*dt)*qfun(a1)
a2=f2*sqrt(dt/(4.d0*D))
N2=rho2*sqrt(D*dt)*qfun(a2)
  print*,'N1',N1
  print*,'N2',N2
```

```
!Creation of a LIBRARY with the y-equispaced nodes of the distribution
!function(vFy0) and the corresponding x nodes (xre) at position 0:

qa1=-a1*erfc(a1)+exp(-(a1**2))/sqrt(pi) !q(a) at 0
x0=5.3E-5 !Initial guess
y=0.d0
!We know that the first components of vFx0 and vFy0 are (0,0)
!to guarantee the convergence of Newton's method
vFx0(0)=0.d0
vFy0(0)=0.d0
do i=1,9999
  y=y+1.d0/10000.d0 !y equispaced
  xn=newton(y,x0,qa1) !It is the argument of qfun (see eq.7)
  xre=sqrt(4.d0*D*dt)*xn+f1*dt !Calculation of the real x
  !Storage of the found values at this iteration
  vFx0(i)=xre
  vFy0(i)=y
  x0=xn !The initial guess of the following iteration will be the previous
  !found value with Newton's method.
enddo

!Writing of the found vectors in a .dat in 2 rows
open (32,file='partentr0n.dat',status='unknown')
do i=0,9999
write (32,*)vFx0(i),vFy0(i)
enddo
close(32)

!Creation of a LIBRARY with the equispaced nodes of the
!distribution function(vFy0) and the corresponding x nodes
!(xre) at position Length.

qa2=-a2*erfc(a2)+exp(-(a2**2))/sqrt(pi) !q(a) at Length
x0=5.3E-5 !Initial initial guess
y=0.d0
!We know that the first components of vFxL and vFyL are (4,0) to
!guarantee the convergence of Newton's method
vFxL(0)=4.d0
vFyL(0)=0.d0
do i=1,9999
  y=y+1.d0/10000.d0
  xn=newton(y,x0,qa2)
  xre=sqrt(4.d0*D*dt)*xn-f2*dt !Changes sign because we are evaluating at
  !Length
  vFxL(i)=Length-xre
  vFyL(i)=y
  x0=xn
enddo

!Writing of the found vectors in a .dat in 2 rows
open (36,file='partentrFn.dat',status='unknown')
write (36,*)vFxL
write(36,*)
write(36,*)vFyL
close(36)
```

```
 do nrealitzacions=1,5000 !Number of realizations of the simulation

 !Counters
 ne0n=0
 neLn=0
 ns0n=0
 nsLn=0

 !Initial particle positions

 nparticulas=50


 !Places the particles in the indicated distribution
 !We use an equispaced initial distribution

 call coloca(nparticulas,Length,xpos)

 !BEGINNING OF THE SIMULATION



   do i=1,itt!Number of temporal iterations

     !Generation of the Gaussian noise using Box-Muller algorithm
     if (nparticulas>0) then
       do j=1,nparticulas
         soroll(j)=r8_normal_ab(0.d0,sqrt(2.d0*kbt*gamma),seed)
       enddo
     else
       print*,'Uep no hi ha particules! i=', i
     endif

     !Integration of the stochastic differential equation using Euler's
     !method
     if (nparticulas>0) then
       do j=1,nparticulas
          xpos(j)=xpos(j)+sqrt(dt)*(soroll(j)/gamma)-
&          dt*derpot(xpos(j))/gamma
       enddo
     endif

     !Generation of two random numbers in a uniform distribution [0,1]
     U1=r8_uniform_01(seed)
     U2=r8_uniform_01(seed)


     !Particle ENTRY from x=0
     if (U1<N1) then

       nparticulas=nparticulas+1
       ne0=ne0+1
       ne0n=ne0n+1

       !Each time that enters a particle, this counter increases by 1
```

```
  U1=r8_uniform_01(seed) !Xi. Different number than before to avoid
  !correlation


  !Calculation of the position where it enters:
  if (U1<=0.9999) then !This is the maximum possible value of the
  !library
    nm=floor(U1*10000)!10000 because the Newton library has 10000
    !components
    m=(vFx0(nm+1)-vFx0(nm))/(vFy0(nm+1)-vFy0(nm)) !Slope^-1
    xentrada=vFx0(nm)+m*(U1-vFy0(nm)) !Position where the particle
    !enters
    xpos(nparticulas)=xentrada

  else !Approximation
    dum=log(2.d0*sqrt(pi)*qfun(a1)*(1.d0-U1))
    xentrada=f1*dt+sqrt(-4.d0*gamma*kbt*dt*dum)
    xpos(nparticulas)=xentrada

  endif

endif


!Particle ENTRY from x=4:
if (U2<N2) then

  nparticulas=nparticulas+1
  neF=neF+1 !Each time that enters a particle, this counter increases
  ! by 1
  neLn=neLn+1

  U2=r8_uniform_01(seed) !Xi. Different number than before to avoid
  !correlation

  if (U2<=0.9999) then !This is the maximum possible value of the
  !library
    nm=floor(U2*10000.d0)!10000 because the Newton library has 10000
    !components
    m=(vFxL(nm+1)-vFxL(nm))/(vFyL(nm+1)-vFyL(nm))
    xentrada=vFxL(nm)+m*(U2-vFyL(nm))
    xpos(nparticulas)=xentrada

  else
    dum=log(2.d0*sqrt(pi)*qfun(a2)*(1.d0-U2))
    xentrada=Length-(-f2*dt+sqrt(-4.d0*gamma*kbt*dt*dum))
    xpos(nparticulas)=xentrada

  endif
endif


!Particle EXIT through the boundaries:
if (nparticulas>0) then
  do j=1,nparticulas
```

```
35              continue
        if (xpos(j)>=Length) then !Exit through the right boundary

          xpos(j)=xpos(nparticulas) !Rewrites the position of the particle
          !that has left for the position of the last particle of the
          !array
          xpos(nparticulas)=0.d0
          nparticulas=nparticulas-1
          nsF=nsF+1 !Each time that a particle exits, this counter
          !increases by 1
          nsLn=nsLn+1
          goto 35
        else if (xpos(j)<0.d0) then !Exit through the left boundary
          xpos(j)=xpos(nparticulas) !Rewrites the position of the
          !particle
          !that has left for the position of the last particle of the
          !array
          xpos(nparticulas)=0.d0
          nparticulas=nparticulas-1
          ns0=ns0+1 !Each time that a particle exits, this counter
          !increases by 1
          ns0n=ns0n+1
          goto 35
        endif

      enddo
    endif

    !DENSITY calculation at the selected times:

    select case (i)
      case (it1)
        do ipos=0,numbin-1
           nbin(ipos)=0
        end do

        do j=1,nparticulas
           ipos=xpos(j)/deltax
           nbin(ipos)=nbin(ipos)+1
        end do

        do ipos=0,numbin-1
           density1(ipos)=nbin(ipos)/deltax
           x=deltax*(0.5+float(ipos))
        end do
      case(it2)
        do ipos=0,numbin-1
           nbin(ipos)=0
        end do

        do j=1,nparticulas
           ipos=xpos(j)/deltax
           nbin(ipos)=nbin(ipos)+1
        end do

        do ipos=0,numbin-1
```

```
            density2(ipos)=nbin(ipos)/deltax
            x=deltax*(0.5+float(ipos))
        end do
      case(it3)
        do ipos=0,numbin-1
            nbin(ipos)=0
        end do

        do j=1,nparticulas
            ipos=xpos(j)/deltax
            nbin(ipos)=nbin(ipos)+1
        end do

        do ipos=0,numbin-1
            density3(ipos)=nbin(ipos)/deltax
            x=deltax*(0.5+float(ipos))
        end do
      case(it4)
        do ipos=0,numbin-1
            nbin(ipos)=0
        end do

        do j=1,nparticulas
            ipos=xpos(j)/deltax
            nbin(ipos)=nbin(ipos)+1
        end do

        do ipos=0,numbin-1
            density4(ipos)=nbin(ipos)/deltax
            x=deltax*(0.5+float(ipos))
        end do
      case(itt)
        do ipos=0,numbin-1
            nbin(ipos)=0
        end do

        do j=1,nparticulas
            ipos=xpos(j)/deltax
            nbin(ipos)=nbin(ipos)+1
        end do

        do ipos=0,numbin-1
            density5(ipos)=nbin(ipos)/deltax
            x=deltax*(0.5+float(ipos))
        end do
      case default
        continue
    end select


enddo !End of the time loop


 print*,'n', nrealitzacions

 !Sum of all the previous density realizations with the most recent one
```

```
     !to perform an average later
    do ipos=0,numbin-1
        densityt1(ipos)=densityt1(ipos)+density1(ipos)
        densityt2(ipos)=densityt2(ipos)+density2(ipos)
        densityt3(ipos)=densityt3(ipos)+density3(ipos)
        densityt4(ipos)=densityt4(ipos)+density4(ipos)
        densityt5(ipos)=densityt5(ipos)+density5(ipos)
    end do

    print*,'ne00',ne0n
    print*,'neLn',neLn
    print*,'ns0n',ns0n
    print*,'nsLn',nsLn

  enddo !End of the realizations loop

  !Writing the averaged densities on an external file:
  open(43,file='density0pot.dat',status='unknown')

    !Average densities at each time:
    do ipos=0,numbin-1
      densityt1(ipos)=densityt1(ipos)/nrealitzacions
      densityt2(ipos)=densityt2(ipos)/nrealitzacions
      densityt3(ipos)=densityt3(ipos)/nrealitzacions
      densityt4(ipos)=densityt4(ipos)/nrealitzacions
      densityt5(ipos)=densityt5(ipos)/nrealitzacions


      write(43,*)deltax*(0.5+float(ipos)),densityt1(ipos),
&    densityt2(ipos),densityt3(ipos),densityt4(ipos),densityt5(ipos)
    end do

    close(43)


    !Printing of the counters:
    print*, 'nparticulas␣final', nparticulas
    print*, 'Han␣entrat␣per␣l"esquerra:', ne0
    print*, 'Han␣entrat␣per␣la␣dreta:', neF
    print*, 'Han␣sortit␣per␣l"esquerra:', ns0
    print*, 'Han␣sortit␣per␣la␣dreta:', nsF


  end program


  !FUNCTIONS AND SUBROUTINES

  !Distribution function F(x)
  function Fx(x,qa)
    implicit none
    real*8 :: x, qa
    real*8 :: qfun
    real*8 :: q, Fx
    q=qfun(x) !x is all the argument (see eq. 7)
    Fx=1.d0-(q/qa)
```

```
  return
end

!q(x)
function qfun(x)
  real*8 :: x, qfun, pi
  common pi, seed
  qfun=-x*erfc(x)+exp(-(x**2))/sqrt(pi)
  return
end

!Function to apply Newton's method to ONLY Fx:
function newton(y,x0,qa)
  real*8 :: x0, x1, tol, tolmax, qa, Fx
  real*8 ::  dx, derx, newton, y
  integer*4 :: it, itmax
  tolmax=1.0E-12
  dx=1.0E-8
  itmax=1000
  it=1
  tol=10

  do while(tol>tolmax.and.it<itmax)
    derx=(Fx(x0+dx,qa)-Fx(x0,qa))/dx !Numerical derivative
    x1=x0-(Fx(x0,qa)-y)/derx !Fx(x0,qa)-y is the function to perform
    !the newton
    it=it+1
    tol=abs(x1-x0)
    x0=x1
  enddo

  newton=x0
  return
end

!Potential profile in the channel:
function pot(x)
  real*8 :: x, pot, kbt, Length
  !kbt=25.d0
  !Length=4.d0
  !pot=-50*x+(8*25)*exp(-(x-2)**2/(2*(0.25**2)))
  !pot=(8*kbt/Length)*x
  pot=0
  return
end

!Calculation of the numerical derivative of the potential
function derpot(x)
  real*8 :: x, derpot, dx, pot
  !dx=1E-8
  !derpot=(pot(x+dx)-pot(x-dx))/(2*dx) !Notice that it can fail when we
  !have a non-derivable potential profile
  return
end

!Congruential generator of a uniformly distributed random number
```

```
  function r8_uniform_01 ( seed )
    integer ( kind = 8 ) k
    real ( kind = 8 ) r8_uniform_01
    integer ( kind = 4 ) seed
    k = seed / 127773
    seed = 16807 * ( seed - k * 127773 ) - k * 2836
    if ( seed < 0 ) then
      seed = seed + 2147483647
    end if
    r8_uniform_01 = real ( seed, kind = 8 ) * 4.656612875D-10
    return
  end

  !Congruential generator of a normal(a,b) distributed random number
  !using the Box-Muller algorithm
  function r8_normal_ab ( a, b, seed)
    real ( kind = 8 ) a
    real ( kind = 8 ) b
    real ( kind = 8 ) r1
    real ( kind = 8 ) r2
    real ( kind = 8 ) r8_normal_ab
    real ( kind = 8 ), parameter :: r8_pi = 3.141592653589793D+00
    real ( kind = 8 ) r8_uniform_01
    integer ( kind = 4 ) seed
    real ( kind = 8 ) x
    r1 = r8_uniform_01 ( seed )
    r2 = r8_uniform_01 ( seed )
    x = sqrt (-2.0D+00 * log ( r1 ) ) * cos ( 2.0D+00 * r8_pi * r2)
    r8_normal_ab = a + b * x
    return
  end

  !Arrangement of the particles at t=0
  subroutine coloca(nparticulas,Length,xpos)
    real*8, dimension(*):: xpos
    integer*8 seed
    real*8 Length
    common seed
    do i=1,nparticulas
      xpos(i)=float(i)*Length/nparticulas !equispaced
    end do
    return
  end
```

Appendix B Code for the numerical solving of Fokker-Planck equation

```
program fokkerplanck

  integer*8 :: N,Nt,nparticulas, kk, it, i
  real*8 :: gamma, Length, kbt, dx, D, dt, deltax, derpot,b
  parameter(N=1000)
  real*8, dimension(0:N+1) :: rho, f
  real*8, dimension(1:N) :: pd, sd, x
  gamma=1000.d0
  Length=4.d0
```

```
kbt=25.d0
deltax=Length/N
D=kbt/gamma
dt=2E-6
Nt=75000000
nparticulas=50
rho(0)=10.d0
rho(N+1)=1.d0


do i=1,N !Initial densities calculation

    x(i)=deltax*(float(i)-0.5)
   rho(i)=nparticulas/Length

end do

!Beginning

do i=0,N+1

   b=deltax*(float(i)-0.5)
   f(i)=-derpot(b)/gamma

enddo


open(78,file='fp.dat',status='unknown')
!Temporal loop

do it=1,Nt

  do i=1,N !Derivatives calculations

     !First derivative:
       pd(i)=-(f(i+1)*rho(i+1)-f(i-1)*rho(i-1))/(2.d0*deltax)

     !Second derivative
      sd(i)=D*(rho(i+1)+rho(i-1)-2.d0*rho(i))/(deltax**2)

  enddo

  do i=1,N !Euler step:

    rho(i)=rho(i)+dt*(pd(i)+sd(i))

  enddo

  !Writing the data at the same time of our simulation:

  if(mod(it,Nt/5)==0)then
    do kk=1,N
      write(78,*)x(kk),rho(kk)
    enddo
    print*,'n',it
    write(78,*)
```

```
  endif

enddo
close(78)

end program

function pot(x) !Different potentials used:
  real*8 :: x, pot, kbt, Length
  kbt=25.d0
  Length=4.d0
  pot=8.d0*kbt/Length*x
  !pot=-50.d0*x+200.d0*exp(-(x-2.d0)**2/(2.d0*(0.25**2)))
  !pot=0.d0
  return
end

!Calculation of the numerical derivative of the potential:
function derpot(x)
  real*8 :: x, derpot, dx, pot
  !dx=1E-8
  derpot=(pot(x+dx)-pot(x-dx))/(2*dx) !Notice that it can fail when we
  !have a non-derivable potential profile
  return
end
```